Delphi 1 / Delphi 2 / ReportSmith 2.5 / ReportSmith 3.0

By *Mark Ostroff*

Leveraging ReportSmith: Part I

Advanced Delphi Reporting

ReportSmith is the Rodney Dangerfield of Delphi. And just as Mr Dangerfield is actually a very savvy performer, ReportSmith is an extremely powerful part of Delphi. ReportSmith can accomplish many reporting tasks that no other reporting tool can match. The key is in knowing how to leverage ReportSmith to its best advantage in your Delphi applications.

Why Have Multiple Report Writers?

To complicate matters, Delphi 2 features two reporting tools: ReportSmith, and a set of native VCL components named QuickReport. It would seem to be redundant to have multiple ways of creating reports. There are good reasons, however, for including both ReportSmith and QuickReport in the current version of Delphi. The challenge is to know when to use each.

QuickReport: simple, fast reporting. QuickReport is designed for applications where simple reporting is needed, and the reporting facility needs to be a part of the application's .EXE. QuickReport follows the Delphi development model very closely. It is, however, a design-time, developer-only solution. As such, QuickReport is best suited for applications where all reporting needs are limited to a specified set of reports, and those reports must be controlled from within your application.

This internal reporting model provides for easier distribution. It also supplies a faster report start-up, since the Windows startup code has already executed by the time your user runs a report. QuickReport's smaller overhead results in reports that generally run faster as well. Ultimately, applications that use QuickReport will have a total disk space requirement that is significantly smaller than what would be needed for using the Delphi/ReportSmith combination. Total system resource requirements are also less when using QuickReport. And, of course, since QuickReport components are Delphi VCL components, you can sub-class them to add your own reporting functionality.

QuickReport does have a few limitations. Live data display during report design in QuickReport is restricted to using a report preview. This requirement to switch between design mode and preview mode makes the design process more complex. In addition, certain reporting needs can't be accomplished with QuickReport. For example, leveraging server resources for large data sets, creating stand-alone reports, and inclusion of crosstabs and data graphs are simply not within the capacity of QuickReport.

ReportSmith: robust, full-featured reporting. ReportSmith, on the other hand, is the power choice. It is well suited for creating complex reports that require the inclusion of data graphs, multiple crosstabs, live data pivoting, drill-down selection criteria, or other ad hoc analysis capabilities. ReportSmith can also be used to build "smart" reports. A single report design can serve double duty. It can be run either within the context of a

Delphi program or as a separate stand-alone report. Thus, ReportSmith is the better choice when you have an application that needs to support both data entry/reporting users and report-only users with the same reports.

ReportSmith also offers a flexible method whereby end-users can design their own reports. These new reports can then be run from within your application. Since ReportSmith is designed for this kind of dual role, it has a more flexible design interface. Its WYSIWYG design mode, along with its live data display, makes report design much simpler than with QuickReport. In fact, although both QuickReport and ReportSmith can build mailing label reports, the live data display design interface of ReportSmith makes it a much better choice for the task.

The design of ReportSmith is tuned for client/server applications. Its power and performance are well suited to reporting on very large data sets. The use of ReportSmith also means that your total disk space requirements can be more efficient when you have multiple applications that reference the single installation copy of ReportSmith. The ability to deploy reports using either the ReportSmith Run-Time or a complete copy of ReportSmith means that you can split your user community with even more finely tuned control. You can support users that just need to run pre-defined reports, as well as people who need to run those reports and create their own. This kind of user-community tailoring is simply not possible with QuickReport.

Which Do I Use?

The key to maximizing the use of any tool is to know when to use it, and how to apply it properly. Some guidelines are given here.

When to use QuickReport. Are load-time performance and distribution size the overriding decision factors? If so, QuickReport is the right choice. With smaller data sets, QuickReport reports will also run faster than ReportSmith reports. Another important issue is whether you need to completely control the creation and execution of reports. Since QuickReport reports are compiled directly into your .EXE, the end-user has no way to modify the report, and no way to run it outside of your application. If you need to create a totally closed solution, QuickReport is the way to do it.

QuickReport's better coordination with the Delphi environment is also a consideration. However, this is more of a Delphi 1 development issue. Delphi 2 has much better integration between Delphi and ReportSmith.

When to use ReportSmith. Are flexibility and power the most important reporting features for your application? If so, ReportSmith is the better choice. It is also the only way to build an open-ended solution. ReportSmith is the only way to create "smart" double-duty reports from a single design. If allowing end-users to create their own reports is a system requirement, you will need to include ReportSmith as part of your solution. Otherwise, developers will be constantly asked to build new reports as users think of them.

Feature	Description
Delphi Connection type	Allows reports to use the data already buffered in the Delphi application's BDE connection.
ReportSmith API	A programmable API for directly interfacing other tools with ReportSmith. (The API documentation is included on the Delphi 2 CD for editions of Delphi which ship with ReportSmith.)
32-bit performance	Record processing is visibly faster due to 32-bit data processing.

Figure 1: New ReportSmith 3.0 features.

Using ReportSmith 3.0 with Delphi. The Developer and Client/Server editions of Delphi 2 include a new 32-bit version of ReportSmith. ReportSmith 3.0 adds some new capabilities that will affect how you approach the integration of ReportSmith reports into your Delphi applications. These new features are summarized in the table shown in Figure 1.

ReportSmith 3.0 differences that affect Delphi development. Some 16-bit versus 32-bit integration issues also exist. You will need to plan for these differences between ReportSmith 2.5 and 3.0 if you need to support both 16-bit and 32-bit Delphi applications. The table in Figure 2 summarizes them.

The New Delphi Connection

Coordination between ReportSmith 2.5 and Delphi 1 has to be implemented programmatically through the use of report parameters, report variables, and/or DDE. The new 32-bit ReportSmith 3.0 adds the ability to directly use the same data connection as your Delphi 2 applications. This feature is referred to as the Delphi Connection.

For most reports, this ability to use the same data buffer results in much better report performance. It also means that report coordination between Delphi and ReportSmith is greatly simplified.

The specific advantages of using the Delphi Connection are:

- Faster data selection. A large part of the start-up time in a traditional ReportSmith report involves running the report's underlying SQL code. ReportSmith uses this code to select which records to include in the report. With the Delphi Connection, both Delphi and ReportSmith use the same data connection. The records to be reported are already selected by the time the report runs. ReportSmith just needs to format the data with a Delphi Connection report.
- Single development language. Traditional reporting with ReportSmith involves the use of either ReportSmith Basic or standard SQL code to perform any necessary

Feature	Description
ReportSmith Data Dictionary	ReportSmith 2.5 has a Data Dictionary Utility that allows a developer to pre-define table and column aliases, field visibility rights, etc. There is no 32-bit version of this utility. However, the editions of ReportSmith that are included in the Delphi 2 Update release do allow you to *use* Data Dictionaries created in the 16-bit tool when you develop 32-bit reports in ReportSmith 3.0. Delphi 2 includes a Delphi 1 installation directory on the CD. You will have to install the ReportSmith Data Dictionary tool from the Delphi 1 directory to enable this feature in ReportSmith 3.0.
Menu Structures	The text of some menu items has changed from ReportSmith 2.5 to 3.0 to reflect the Win32 standards. For example, 2.5 uses "Character" where 3.0 uses "Font" in the SpeedMenus. (ReportSmith 3.0 currently is the only Windows 95 logo-compliant reporting tool.) These differences may affect program code that relies on specific text in a menu selection when you have to support both 16-bit and 32-bit applications.
Picture Support	Both versions of ReportSmith support the use of .BMP, .PCX, and .DIB graphic formats. Due to a change in the copyright license for certain graphic formats, the 32-bit version no longer supports the .GIF and .TIF formats. If you need to support both 16-bit and 32-bit reporting, you will need to avoid using .GIF or .TIF graphics.

Figure 2: 16-bit features supported differently in ReportSmith 3.0.

code-based operations. This is not a big issue with some people. Others, however, prefer to learn just one language. The Delphi Connection is the way to place total control over data calculations into your Delphi application. Adding the power of DDE and the new ReportSmith API means you can then control virtually everything in ReportSmith with Object Pascal code.

- More powerful calculated fields. The entire data set of a Delphi Connection report is controlled by your Delphi application, including any calculated fields you define in your data set component. While ReportSmith itself has the ability to create calculated fields, the power of these "derived fields" is limited to what you can accomplish with either ReportSmith Basic or SQL code. Delphi can, of course, take advantage of the full power of Object Pascal. You can use your own custom code, commercial statistical packages, and any function call contained in DLL libraries or the Windows API in creating derived fields. There are *no* limits to the calculated fields you can create in Delphi.

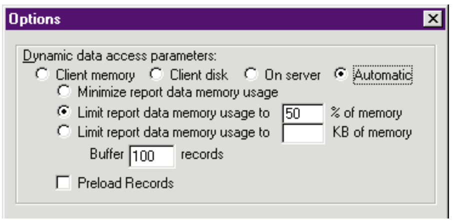
Using Delphi-supplied parameters. The Delphi Connection is only available for building 32-bit reports in ReportSmith 3.0. Using report parameters is, therefore, the only option available for 16-bit applications. There are reasons why you might still want to use this capability — even in 32-bit applications — rather than using the direct Delphi Connection:

- Powerful report control. ReportSmith report variables can be used for a wide variety of functions that would not be possible by simply using the same data connection as your Delphi application. Report variables can be used for internal report processing that reaches outside the current Delphi data set. For example, you may want to highlight in red any purchase orders greater than a certain target amount. Report variables give you the ability to build a report that gets this threshold level at report time by passing a value from a Delphi variable, a value stored in an .INI file or the Windows 95 Registry, or by prompting the user. You can even take advantage of this kind of report variable processing with a report that *does* use the Delphi Connection to select the records to include in the report.
- Creating context-sensitive "smart" reports. Creating reports that can be run by themselves is one of the primary reasons to use ReportSmith. If you also need these reports to respond to the current Delphi environment when run from within your application, report variables provide the ability to create context-sensitive reports. These "smart" reports will feed off the parameters passed by Delphi from within your application. Then, when the same report is run outside of your application, it will automatically prompt the user for the necessary information to run the report in stand-alone mode. (Delphi Connection reports cannot be run by themselves. Your Delphi application must be running to be able to run a Delphi Connection report. Otherwise, the report will auto-exit with an error message.)
- Creating cross-platform reports. Many organizations recognize that the migration from 16-bit to 32-bit Windows platforms will take a while. The applications that support this transition best are the ones that provide for a "develop once, deploy on both" strategy. Since the Delphi Connection is a 32-bit only feature, cross-platform reports must rely on report variables and Delphi *TReport* parameters.
- Reporting on huge data sets. ReportSmith was originally designed as a client/server reporting tool. It provides tuning parameters in the Options dialog box for reporting on very large sets of data (see Figure 3). These Dynamic Data Access features are turned off when the Delphi Connection is used. (All data selection is controlled by the BDE buffer of your Delphi application.) As such, you may want to carefully weigh the performance ramifications of creating Delphi Connection reports when selecting huge data sets.

The New ReportSmith API

ReportSmith 3.0 also ships with the new ReportSmith API. This standard programming interface can be used by Delphi to control ReportSmith directly. You are no longer restricted to using the less powerful DDE route to direct the operation of ReportSmith from within your Delphi applications. The documentation for this API is included on the Delphi 2 CD for editions that include ReportSmith 3.0.

Using ReportSmith with Delphi



Options

Dynamic data access parameters:

- ( ) Client memory ( ) Client disk ( ) On server (•) Automatic
- ( ) Minimize report data memory usage
- (•) Limit report data memory usage to 50 % of memory
- ( ) Limit report data memory usage to KB of memory
- Buffer 100 records
- [ ] Preload Records

Figure 3: Dynamic Data Access settings.

As noted earlier, two main methods exist for integrating ReportSmith reports into your Delphi applications. The first method, using Delphi-supplied report parameters, applies to both 16-bit and 32-bit versions. The second method is to use the new 32-bit Delphi Connection.

Using Delphi-supplied report parameters. The key to using ReportSmith is in coordinating two separate executables, your Delphi applications and ReportSmith. All versions of ReportSmith support the use of report variables. Delphi's *TReport* component supports a mechanism whereby you can pass values from your Delphi application into report variables contained within your report.

The basic development sequence for using Delphi-supplied report parameters is as follows:

- Build a ReportSmith report that uses report variables. Define report variables for any of the items you want your Delphi application to use for controlling the report's execution.
- Add one or more *TReport* components to your Delphi application to control the report execution.
- Add Delphi code to your application to pass the appropriate values to the report's report variables when the reports are run or refreshed.

Building a report with report variables. ReportSmith allows you to create variables within the report design that are not tied to the data. Instead, these report variables have values that are typically supplied by the user and then become available within the report for logic tasks such as record selection. For example, you can create a report variable to store a user-supplied state. This value can then be used in a selection criteria to limit the report to those records from the state the user entered into the report variable. In the case of executing these reports from a Delphi application, your application becomes the report's user, and can supply the necessary values for any report variables defined in the report.

"Smart" reports. Report variables thus become the mechanism whereby you can create "smart" reports. The values for report variables can be supplied directly by the end-user, or indirectly by passing them from a Delphi application. When the report is run from the Delphi application, ReportSmith verifies if all report variables have already been supplied values. If they have, the report proceeds and automatically uses the values supplied by Delphi. If any of the report variables have not been assigned a value, or if the report is run "stand-alone" (i.e. outside of any Delphi application), ReportSmith is intelligent enough to prompt the end-user to supply values directly.

Start ReportSmith and create a new columnar report based on the CUSTOMER.DB table in the DBDEMOS alias. (This alias is created when you install Delphi; it points to Delphi's sample data directory.) You will use this report as the basis for building your first "smart" report.

Defining a report variable. Use the **Tools | Report Variables** menu selection to bring up the Report Variables definition dialog box. This is where you will create custom prompts and definition specifications for your report variable. ReportSmith will use the information to create a dialog box for prompting the user for each report variable.

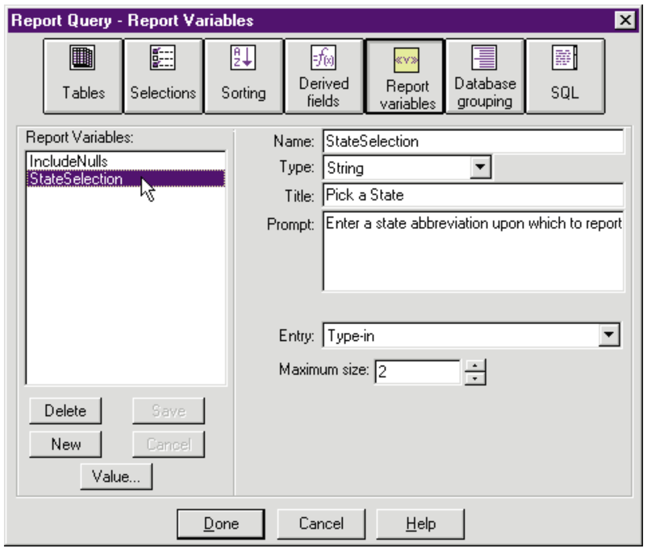
- The **Name** field is where you give your report variable its designated name. This name will appear throughout drop-down lists in ReportSmith wherever a report variable can be used.
- The **Type** of a report variable can be defined as string, number, date, time, or date/time. The variable type will affect the various entry options that will appear below.
- The **Title** refers to the title string that will be displayed in the dialog box ReportSmith uses to prompt the user for a value to assign this report variable.
- The **Prompt** value refers to the string that will appear as the message within the prompting dialog box.

For each report variable, you have a choice of **Entry** options. The report can have the user supply a value by typing it in, choosing from a defined list, choosing the value from a table, or choosing between two values (yes/no, true/false, etc.). When you select **Choose value from a table**, you can use a different data source type than the rest of your report. Various option settings will appear based on the type of the report variable and the entry option chosen.

To save the definition of your new report variable, click the **Add** button. Just editing the current data on the right side of the dialog box will change the currently highlighted report variable, not create a new one. To create a new report variable, you must click on the **New** button first. Create the *StateSelection* report variable that is shown in Figure 4.

When you are finished creating report variables, ReportSmith will prompt you to supply initial values for each variable. However, you won't see any effect on your report; you haven't told ReportSmith how to use your new report variable yet.

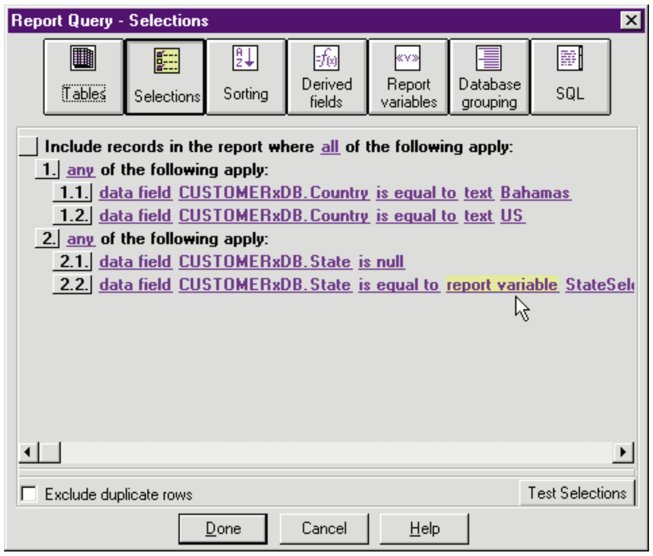
Using a report variable. Report variables are most often used to provide run-time record selection. In the previous example, say you wanted to make the selection of which state to include — something the user determines at run time. Once you have the *StateSelection* report variable defined, you can add it to the record selection criteria. Just use the drop-down lists in the Selections dialog box to switch from a "hard coded" text value to use your report variable instead. Say your report originally limited the records to those with a country of either the Bahamas or the US. Figure 5 shows how your selection criteria might look after adding the *StateSelection* report variable to control which US state gets included.



Report Query - Report Variables

Report Variables:
IncludeNulls
StateSelection

Name:	StateSelection
Type:	String
Title:	Pick a State
Prompt:	Enter a state abbreviation upon which to report
Entry:	Type-in
Maximum size:	2



Report Query - Selections

```
Include records in the report where all of the following apply:
 1.  any of the following apply:
   1.1.  data field CUSTOMERxDB.Country is equal to text Bahamas
   1.2.  data field CUSTOMERxDB.Country is equal to text US
 2.  any of the following apply:
   2.1.  data field CUSTOMERxDB.State is null
   2.2.  data field CUSTOMERxDB.State is equal to report variable StateSel
```

Exclude duplicate rows

Figure 4 (Top): The Report Variables definition dialog box.
Figure 5 (Bottom): Using a report variable for record selection.

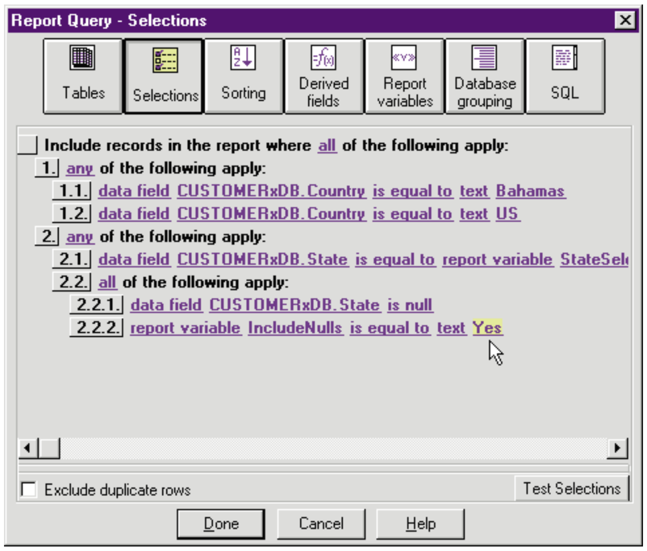
You can also make use of selection lists in conjunction with report variables to create some pretty complex criteria. Say, for example, that you only want null state records included if the user wants to include them. To accomplish this task, first create a *NullSelection* report variable. The parameters should appear as shown in Figure 6.

Once this report variable is defined, open the Selections dialog box and click in the box at the left side of the criteria that says the State field can be null. A pop-up list will appear. Pick the **create a new list and move this item into it** option. Now add a new criteria in that list that states that the *IncludeNull* report variable must be equal to the text "Yes". Your criteria should now look like the dialog box shown in Figure 7.

Now you should be prompted twice, once for the state to select and once to decide whether to include records with a null value in the State field. Selecting **Yes** to the second prompt will cause the records from the Bahamas to appear. Selecting **No** will limit the report to records from the US.

Parameter	Value to Enter
Name	Include Nulls
Type	String
Title	Include Nulls?
Prompt	Should records with no state be included?
Entry	Choose between two values
Message or Dialog	Message
Yes value	Yes
No value	No

Figure 6: Defining a Yes/No report variable.



Report Query - Selections

```
Include records in the report where all of the following apply:
 1.  any of the following apply:
   1.1.  data field CUSTOMERxDB.Country is equal to text Bahamas
   1.2.  data field CUSTOMERxDB.Country is equal to text US
 2.  any of the following apply:
   2.1.  data field CUSTOMERxDB.State is equal to report variable StateSel
   2.2.  all of the following apply:
     2.2.1.  data field CUSTOMERxDB.State is null
     2.2.2.  report variable IncludeNulls is equal to text Yes
```

Exclude duplicate rows

Figure 7: Expanding the selection criteria with a list.

Changing the value of a report variable. You may want to test multiple parts of a report controlled by a report variable. You might want to print a different set of records without having to reload the report. In either case, you need to alter the value initially assigned to a report variable. Simply select **Tools | Report Variables** from the main menu, click on the name of the report variable whose value you want to change, and click on the **Value** button that appears near the bottom of the dialog box. ReportSmith will prompt you for a different value in the same manner it prompted for the initial value.

Creating the Delphi Application

The next step in creating ReportSmith reports that can run from your Delphi application is to build the necessary capabilities into your Delphi application. This involves the use of Delphi's *TReport* component.

The *TReport* component. The *TReport* component resides in the Data Access page of Delphi's Component Palette. It is what you will use to control all aspects of running ReportSmith reports. While other properties exist, the table in Figure 8 shows the key properties upon which most of your development attention will be focused. Please note the *LaunchType* property is new to Delphi 2.

The *TReport* component also supplies several methods for dealing with ReportSmith from within your Delphi application. The key methods are listed in Figure 9. All *TReport* methods operate via DDE, but you do not need to pro-

Property	Function
AutoUnload	Determines whether ReportSmith exits or stays in memory once the report is finished. Keeping ReportSmith resident can significantly increase the performance of applications that print multiple reports. Use the *CloseApplication* method to shut down ReportSmith if *AutoUnload* is set to *False*.
InitialValues	Use this array of *TStrings* to specify initial values for any report variables included in the report. See the discussion of passing report variable values for complete details.
LaunchType	Delphi 2 only. Determines whether ReportSmith or the ReportSmith RunTime will be launched. The default value of *ltDefault* will launch the ReportSmith 3.0 design environment when you double-click on the *TReport* in Delphi's IDE, and will execute the ReportSmith 3.0 RunTime when you launch a report from within a running Delphi application.
Preview	Determines whether the *Run* method launches a print preview or hard copy output. If *Preview* is set to *True*, use the *Print* method instead of *Run* to force hard copy output.
ReportDir	The directory where the report resides. At run time, use the function `ExtractFilePath(ParamStr(0)` if you want to set this property to the current application directory. Do *not* leave this property blank.
ReportName	The name of the .RPT file to run. Place *only* the name of the file here. All directory information should be placed in the *ReportDir* property. Any directory information placed in the *ReportName* property will be ignored.

Figure 8: Key properties of the *TReport* component.

Method	Function
CloseApplication	Tells ReportSmith to perform a **File | Exit**. Use this method when your *TReport*'s *AutoUnload* property is *False*. Make sure you call *CloseApplication* before exiting your Delphi 1 applications. In Delphi 1, ReportSmith will not exit even when your application shuts down. This behavior has changed with 32-bit applications. ReportSmith will automatically shut down when you exit a Delphi 2 application, even if *AutoUnload* is *False*.
Connect	Allows you to connect to a database and bypass the ReportSmith database login. The *Connect* method is not needed for reports that use the Delphi Connection.
Print	Forces a hard copy output of the report, regardless of the value of the *Preview* property.
RecalcReport	Tells ReportSmith to refresh the report. Use this method whenever you change the value of a report variable or navigate to a new record set in a Delphi Connection report.
Run	Runs the designated report, looking at the value of the *Preview* property to determine if a preview or a hard copy should be made.
RunMacro	Lets you run a ReportBasic macro.
SetVariable	Lets you set or change the value of a report variable. If you specify a report variable name that is not used in the current report, the variable will simply be ignored.
SetVariableLines	Similar to *SetVariable*, this method lets you set or change the value of a report variable by using an array of type *TStrings*.

Figure 9: Key *TReport* component methods.

gram any of the DDE operations yourself. They are automatically encapsulated within the *TReport* component. You also do not need to use any of the Delphi DDE components when you work with *TReport*.

All *TReport* methods provide a True/False return value. This return value gives you feedback on the success or failure of *TReport*'s action. If the method's action gets a "command received" confirmation from ReportSmith, the return value will be *True*. A return value of *False* indicates some kind of problem with the DDE conversation.

Integrating ReportSmith report variables into your Delphi application. Delphi applications can control ReportSmith by pre-defining the values of any report variables used. Set up these report variable values in any way you want within Delphi. For example, you can even create a "Query-by-Form" interface to control all the specifics of a report, assign any values the end-user has set to the appropriate report variables, then launch the report. Any report variable you supply a value for will be skipped in ReportSmith's prompting process. ReportSmith will still prompt the user for any report variables that do not yet have a value.

Keep in mind that Delphi deals with report variables in two ways, depending on whether the report is already open. If the report hasn't been opened, you need to assign values by setting up the *InitialValues* property of the *TReport*. Once the report is running, set or change the values of report variables using the *SetVariable* or *SetVariableLines* method.

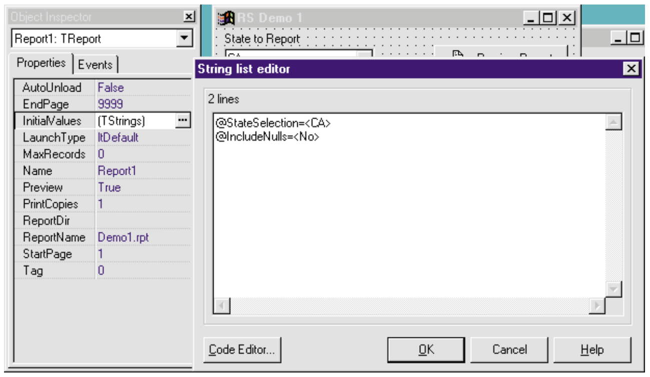
Do not use *SetVariable* or *SetVariableLines* without an open report. These methods try to establish a DDE conversation with ReportSmith. If ReportSmith isn't running, these methods will launch it. Your *TReport* may not have all its properties set properly for your report, so ReportSmith will not be launched with the proper setup. You might not even have a report open, in which case your report variable changes will be ignored with no way for Delphi to know they have been lost. Make sure you use the *InitialValues* property to establish report variable values before launching a report.

Assigning initial values to report variables. You can assign initial values to report variables either at design time using the Delphi Object Inspector, or in application code. To set initial values at design time, double-click on the edit box for the *InitialValues* property and enter the report variable name and its value in the string editor.

Notice that the proper syntax for each entry in the *InitialValues* property is:

```
@ReportVarName=<ValueToAssign>
```

No spaces are allowed. You also need to remember that report variable names are case-sensitive. You must duplicate the report variable's name *exactly* as it was defined in ReportSmith. In the example shown in Figure 10, enter-


Object Inspector

Report1: TReport

Properties | Events

AutoUnload	False
EndPage	9999
InitialValues	(TStrings)
LaunchType	ltDefault
MaxRecords	0
Name	Report1
Preview	True
PrintCopies	1
ReportDir	
ReportName	Demo1.rpt
StartPage	1
Tag	0

RS Demo 1

State to Report

String list editor

2 lines

```
@StateSelection=<CA>
@IncludeNulls=<No>
```

Figure 10: Assigning initial report variable values at design time.

ing a variable name of *StateSelection* would result in the value being ignored. ReportSmith's own prompting logic would then take over and ask the user to select a state. So, if you think your Delphi application is supplying a value to a report variable but the report still prompts for it, check that you are using the correct report variable name.

You can also attach code to a Delphi event that sets up the initial values. Use the *Add* method of the *InitialValues* property to set initial values into a report variable. For example, you might place the following code on a **Print** button to initialize a report and then launch it:

```
procedure TForm1.BtnPreviewClick(Sender: TObject);
begin
  { Set to current application directory }
  Report1.ReportDir  := ExtractFilePath(ParamStr(0));
  Report1.ReportName := 'Demo1.rpt';
  Report1.InitialValues.Add('@StateSelection=<CA>');
  Report1.InitialValues.Add('@IncludeNulls=<No>');
  Report1.run;
end;
```

Changing the value of a report variable. Once a report is opened, you must change report variables in a way that can be recognized by a running report. Delphi uses *TReport* methods to change report variables via a two-way DDE conversation. Changing a report variable does not automatically refresh the report based on the new value.

This separation allows you to set multiple new values before using the *RecalcReport* method to force a refresh of the report. You should usually test the success of any changes before trying to tell ReportSmith to recalculate the report. Here's an example:

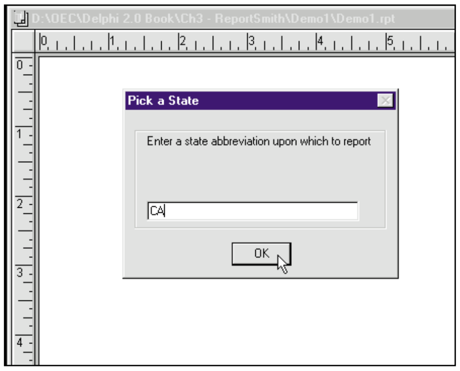
```
if Report1.SetVariable('IncludeNulls','Yes') then
  Report1.RecalcReport;
```

It is extremely important to note the syntax used by the *SetVariable* and *SetVariableLines* methods is very different from what you use to set the *InitialValues* property.

An Example Delphi Application

The RS_DEMO1 project gives you an example of how to control a ReportSmith report from within a Delphi application. This project makes use of the techniques described earlier to control the running of the DEMO1.RPT report file via report variables.

DEMO1.RPT is an example of a "smart" report. First, start up ReportSmith and open the DEMO1.RPT report file by itself. You will see that the report automatically prompts you to supply a state abbreviation (see Figure 11). It then asks if you want to include records with a blank in the state field. Once you answer these questions, the report proceeds.


D:\OEC\Delphi 2.0 Book\Ch3 - ReportSmith\Demo1\Demo1.rpt

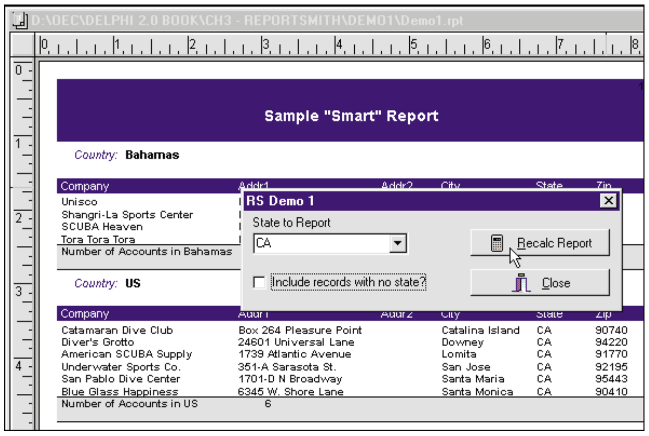
Pick a State

Enter a state abbreviation upon which to report

CA

Figure 11: DEMO1.RPT running on its own.

Now try running the Delphi application RS_DEMO1.EXE and see the difference in running the same report controlled by this Delphi application (see Figure 12). Since Delphi is supplying the values for the two report variables, the report already has all the information it needs before it starts. The end-user is never prompted by ReportSmith.


D:\OEC\DELPHI 2.0 BOOK\CH3 - REPORTSMITH\DEMO1\Demo1.rpt

Sample "Smart" Report

Country: Bahamas

Company	Addr1	Addr2	City	State	Zip
Unisco					
Shangri-La Sports Center					
SCUBA Heaven					
Tora Tora Tora					

Number of Accounts in Bahamas

Country: US

Company	Addr1	Addr2	City	State	Zip
Catamaran Dive Club	Box 264 Pleasure Point		Catalina Island	CA	90740
Diver's Grotto	24601 Universal Lane		Downey	CA	94220
American SCUBA Supply	1739 Atlantic Avenue		Lomita	CA	91770
Underwater Sports Co.	351-A Sarasota St.		San Jose	CA	92195
San Pablo Dive Center	1701-D N Broadway		Santa Maria	CA	95443
Blue Glass Happiness	6345 W. Shore Lane		Santa Monica	CA	90410

Number of Accounts in US 6

RS Demo 1

State to Report: CA

☐ Include records with no state?

Figure 12: DEMO1.RPT run by a Delphi application.

The main code that runs the RS_DEMO1 application is listed in Figure 13. Take particular note of the *SetVars* procedure. It tests whether the application has run ReportSmith before deciding on how to implement changes made to the items supplying values to the report variables. If the report hasn't been started, the code uses the *InitialValues* property. If the report is already running, the *SetVariables* method is used and the **Recalc Report** button is activated. Try changing the value of one or both items in RS_DEMO1 and watch the effect of hitting the **Recalc Report** button. The report will update behind the scenes.

Until Next Month ...

Despite its Rodney Dangerfield reputation, ReportSmith is a powerful reporting tool that can be easily integrated into

```
procedure TForm1.SetVars(Sender: TObject);
begin
  if BtnPreview.Visible then
    begin
      Report1.InitialValues.Add(
        '@StateSelection=<'+cbStateList.Text+'>');
      if CbxNulls.checked then
        Report1.InitialValues.Add('@IncludeNulls=<Yes>')
      else
        Report1.InitialValues.Add('@IncludeNulls=<No>');
    end
  else
    begin
      Report1.SetVariable('StateSelection',
                         cbStateList.Text);
      if CbxNulls.checked then
        Report1.SetVariable('IncludeNulls','Yes')
      else
        Report1.SetVariable('IncludeNulls','No');
      BtnRecalc.Enabled := True;
    end;
end;

procedure TForm1.BtnRecalcClick(Sender: TObject);
begin
  Report1.RecalcReport;
  BtnRecalc.Enabled := False;
end;

procedure TForm1.BtnPreviewClick(Sender: TObject);
begin
  Report1.ReportDir   := ExtractFilePath(ParamStr(0));
  Report1.Run;
  BtnRecalc.Top       := 16;
  BtnPreview.Visible  := False;
  BtnRecalc.Visible   := True;
end;
```

Figure 13: The main code that runs the RS_DEMO1 application.

your Delphi applications. Hopefully this article has shown you a few techniques you can use in your own applications. This month, we tackled the issues of when to use ReportSmith, and how to call ReportSmith reports using report variables. Next month we'll address the use of the new Delphi Connection. ∆

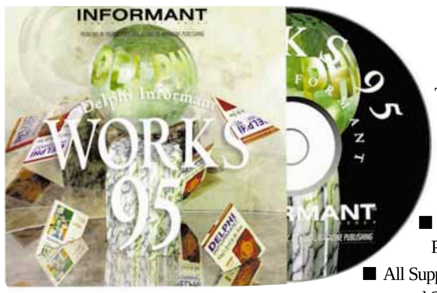
Some of the material in this article is excerpted by permission from the author's chapter on ReportSmith in the book "Delphi In Depth", copyright 1996 by Osborne/McGraw-Hill. All code examples given here are found on the CD that accompanies the book.

The demonstration project and report referenced in this article are available on the Delphi Works CD located in INFORM\96\AUG\DI9608MO.

Mark Ostroff has over 16 years experience in the computer industry. He began by programming mini-computer medical research data acquisition systems, device interfaces, and process control database systems in a variety of 3GL computer languages. He then moved to PC's using dBASE and Clipper to create systems for the US Navy and IBM's COS Division. He volunteered to helped create the original Paradox-based "InTouch System" for the Friends of the Vietnam Veteran's Memorial. Mark has worked for Borland for the past five years as a Systems Engineer, specializing in database applications.

The Must Have Reference Source For The Serious Delphi™ Developer

The Entire Text of Over 60 Technical Articles Appearing in Delphi™ Informant® in 1995



The Delphi™ Informant® Works 1995 CD-ROM Includes:

- Over 60 Technical Articles
- Text and Keyword Search Capability
- Improved Speed and Performance
- All Supporting Code and Sample Files
- 16-Page Color Booklet
- Electronic Version of *Delphi Power Tools* Catalog
- Electronic Version of *Borland C++ Power Tools* Catalog
- Third-Party Add-In Product Demos
- CompuServe Starter Kit with $15 Usage Credit.

A $105 Value Available Now for only

$39.95

California residents add 7¼% Sales Tax, plus $5 shipping & handling for US orders. (International orders add $15 shipping & handling)

Call Now Toll Free 1-800-88-INFORM

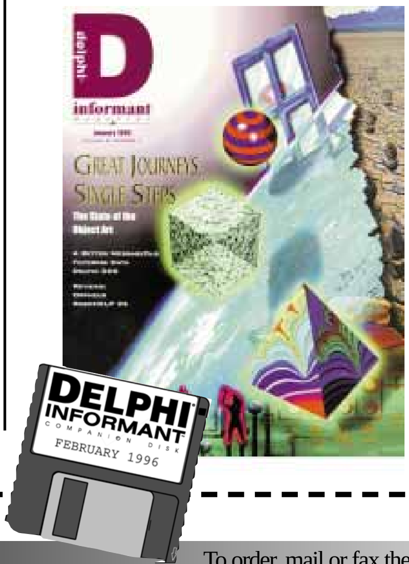
1-800-884-6367 Ask for offer # PI95 To order by mail, send check or Money Order to:
Informant Communications Group, Inc.
ATTN: Works CD offer # PI95
10519 E. Stockton Blvd, Suite 142 Elk Grove, CA 95624-9704
or Fax your order to
916-686-8497

Get Informed!

Subscribe to Delphi Informant, The Complete Monthly Guide to Delphi Development and stay ahead of the rapid application development curve.

Order Now and Get One Issue FREE!

For a limited time you can receive the first issue FREE plus 12 additional issues for only $49.95 That's nearly 25% off the yearly cover price!



Each big issue of *Delphi Informant* is packed with Delphi tips, techniques , news, and more!

- Client/Server Application Development
- Delphi Programming
- Using the Borland Database Engine
- Object -Oriented Programming
- Creating Reusable Components
- Product Reviews
- News from the Delphi Community
- Delphi User Group Information

Payment Method...

- ❑ Check (payable to Informant Communications Group)
- ❑ Purchase Order-- Provide Number ____________
- ❑ Visa
- ❑ Mastercard
- ❑ American Express

Card Number ____________

Expiration Date ____________

Signature ____________

International rates

Magazine-only
$54.95/year to Canada
$74.95/year to Mexico
$79.95/year to all other countries

Magazine AND Companion Disk Subscriptions
$124.95/year to Canada
$154.95/year to Mexico
$179.95 to all other countries

California Residents add 7¼% sales tax on disk subscription

☐ YES

To order, mail or fax the form below or call (916) 686-6610 Fax: (916) 686-8497

I want to sharpen my Delphi Programming skills. I've checked the subscription plan I'm interested in below

☐ Magazine-Only Subscription Plan...
13 Issues Including One Bonus Issue at $49.95.

☐ Magazine AND Companion Disk Subscription Plan...
13 Issues and Disks Including One Bonus Issue and Disk at $119.95
The Delphi Informant Companion Disk contains source code, support files, examples, utilities, samples, and more!

☐ Delphi Informant Works 1995 CD-ROM $39.95
US residents add $5 shipping and handling. International customers add $15 shipping and handling.

Name ____________

Company ____________

Address ____________

City ____________ State ______ Zip Code ______

Country ____________ Phone ____________

FAX ____________ e-Mail ____________

PI95